CSCE 221 Cover Page

Programming Assignment #2

First Name Last Name UIN

User Name E-mail address

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Date

Programming Assignment 2 (100 points)

Due June 16 at midnight to eCampus

Objectives:

- Implement the Minimum Priority Queue (MPQ) ADT based on three data structures: vector, linked list and binary heap.
- Test each MPQ operation for correctness using integer type for data structures.
- Apply MPQ for scheduling computer processes (jobs).

This programming assignment consists of three phases.

1. **(30 points) Phase 1**: Implement the minimum priority queue (MPQ) with the operations: `remove_min()`, `is_empty()`, and `insert()` using two different implementations based on

   (a) vector of type `T`

   (b) linked list of type `T`

   where `T` is template typename parameter. The STL vector and linked list containers are allowed to be used for the implementation of MPQs.

   Test the priority queue data operations for correctness using initially `int` type as `T`, and then in the Phase 3 `T` is `CPU_Job`. In your report provide implementation details of MPQs, the running time in terms of big-O asymptotic notation of each operation, and an evidence of testing the operations.

2. **(20 points) Phase** 2: Implement the binary heap operations: `remove_min()`, `is_empty()`, and `insert()`, and then implement MPQ based on the binary heap of type `T`. The description of the binary heap can be found in the textbook Chapter 6 and on the lecture slides: *"Binary Heap"* and "*Priority Queues*". You can use the STL vector to implement binary heap data structure. Test the binary heap ADT using initially `int` as T, and then `CPU_Jobs` as `T` (in Phase 3). In your report provide implementation details, the running time in terms of big-O asymptotic notation of each operation, and an evidence of testing the operations.

3. **(20 points) Phase** 3: Write an application based on the MPQs implemented in Phase 1 and 2 to simulate the scheduling of computer processes (called also "*jobs*" in operating systems) to run on a single CPU.

   (a) You will use the struct `CPU_Job`.

```
struct CPU_Job {
   int ID;     // process ID
   int length; // process length
   int prior;  // process priority
   CPU_Job(int a=0, int b=0, int c=0) : ID(a), length(b), prior(c) {};
   // compares jobs' priorities and/or IDs
   bool operator<(const CPU_Job& job);
};
```

   (b) Description.

   i. The simulation of CPU running is done by using a loop where one iteration corresponds to the unit of the CPU time called a "*time slice*." The value of the loop index is the current CPU time, and it goes from 1 to a certain integer limit. From among all jobs waiting to be processed in a time slice, the CPU must select the job with highest priority (see the definition of the highest priority below) and run it.

   ii. The jobs to run the CPU are read from an input text file (in the directory `PA2-DataFiles`) where each line represents a job. The format of each line consists of 3 integers: *job ID, length, priority.*

   iii. Job IDs are unique positive integers (they do not need to be consecutive numbers).

   iv. A job length is the time CPU will spend running it (we assume here that we know it). The job length is measured in time slices and, to simplify, is an integer between 1 and 10, inclusive. When the CPU runs a job, its job length is decreasing until it is zero, and then it is removed from the CPU queue and another job is selected.

v. A job priority is represented also by an integer between -20 (highest priority is negative 20) and 19 (lowest priority is 19), inclusive as well. Jobs are scheduled based on their priorities: highest priority jobs will be scheduled first. Pay attention to the fact that the *higher* priority means *lower* numerical value (therefore we can use Minimum PQ). If two or more jobs have the same priority value, break ties by job IDs: a job with a lower ID is scheduled first.
File input format (given):

job ID	length	priority
281	1	12
825	2	5
111	4	19
382	3	-7

vi. We assume that jobs cannot be interrupted – once a job is scheduled on the CPU, it runs for a number of time slices equal to its length.

(c) Your program should create an output text file (possibly with a name based on the input file but it is not required). For each time slice, print one line to this file containing job ID of the job currently running the CPU, its priority, and the remaining time to run (in time slices). With every loop iteration the length of the job running the CPU is decreased by 1. The format of the output line is as follows: "*Job [job ID here] with length n and priority p*". For example:

```
Job 382 with length 3 and priority -7
Job 382 with length 2 and priority -7
Job 382 with length 1 and priority -7
Job 825 with length 2 and priority 5
Job 825 with length 1 and priority 5
Job 281 with length 1 and priority 12
Job 111 with length 4 and priority 19
Job 111 with length 3 and priority 19
Job 111 with length 2 and priority 19
Job 111 with length 1 and priority 19
No more jobs to run
```

If there are no jobs to run, the program issues the output: "*No more jobs to run*", and stops.

(d) Reading data from the input files and writing data into the output files with the required format. There are 5 test input files available in the directory `PA2-DataFiles`. They contain 4, 10, 100, 1000, 10000 and 100000 jobs (input lines), respectively. Run your initial tests on a small input file with less than 11 jobs. And later on test your program on larger files. Your program should run correctly on TA's input.

4. (**10 points**) For large files, get timings for MPQ run on CPU_Job with large number of jobs (of size 1000, 10000, 100000 jobs). There will be 9 timings for your implementations: vector, linked list and binary heap.

To time the MPQ algorithm, you use the function `clock()` and/or the C++ class `chrono`. You need to discuss it in your report and compare them with the theoretical results presented in class.

(a) Here is the example of how to use the function `clock()`.

```
#include <ctime>
//...
clock_t t1, t2;
t1 = clock(); // start
//...
// operations you want to measure the running time
/...
t2 = clock(); // stop
double diff = (double)(t2 - t1)*1000/CLOCKS_PER_SEC;
cout << "Timing: " << diff << " milisec" << endl;
```

(b) You use the STL class `chrono` together with the class `steady_clock`. You can find information about the class chrono on this website:
http://www.cplusplus.com/reference/chrono/
Here is an example how to use it:

```
#include <chrono>
using namespace std;
using namespace chrono;
```

```
// ...
steady_clock::time_point t1 = steady_clock::now(); //start
//...
// operations you want to measure the running time
//...
steady_clock::time_point t2 = steady_clock::now(); //stop
duration<double> time_span = duration_cast<duration<double>>(t2 - t1);
cout << "Timing: " << time_span.count()*1000 << " milisec" << endl;
// ...
```

5. Instructions
   (a) Your C++ files can be named as follows: `main.cpp`, `vector-mpq.h`, `linkedlist-mpq.h`, `binaryheap-mpq.h` and (optional) `cpu-job.h`.
   (b) Compile your program by
   `c++ -std=c++11 *.cpp -o pa2`
   (c) Run your program by executing `./pa2`
   (d) Test your code and collect output data for your report for small sizes (4 and 10 jobs) only.
6. Submission
   (a) Zip the C++ files above and any additional files (input and output files).
   (b) "turnin" your zip file to eCampus.
   (c) (**20 points**) Submit an electronic copy of cover page, report (see below) in lab to your lab TA. Typed report made preferably using "LyX/LaTeX"
      i. (2 pts) Program description; purpose of the assignment.
      ii. (2 pts) Instructions to compile and run your program; input and output specifications.
      iii. (5 pts) Programming style, and program organization and design: naming, indentation, whitespace, comments, declaration, variables and constants, expressions and operators, error handling and reporting, files organization, operators overloading. templates. Please refer to the PPP-style document.
      iv. (5 pts) Discussion of the implementation and running time in the terms of big-O.
      v. (6 pts) Presenting the testing results, use a table to present timings for each implementation (9 timings in total).